**NAME:MANSI JUMDE**

**CLASS/BATCH:A2-B1**

**ROLL NO:09**

**DAA PRACTICAL 5**

**Aim**: Implement a dynamic algorithm for Longest Common Subsequence (LCS) to find the

length and LCS for DNA sequences.

Problem Statement:

(i) DNA sequences can be viewed as strings of A, C, G, and T characters, which

represent nucleotides. Finding the similarities between two DNA sequences are an

important computation performed in bioinformatics.

[Note that a subsequence might not include consecutive elements of the original sequence.]

**TASK 1:** Find the similarity between the given X and Y sequence.

X=AGCCCTAAGGGCTACCTAGCTT

Y= GACAGCCTACAAGCGTTAGCTTG

Output: Cost matrix with all costs and direction, final cost of LCS and the LCS.

Length of LCS=16

Code-

#include <stdio.h>

#include <string.h>

#define MAX 100

void printMatrix(int dp[MAX][MAX], char dir[MAX][MAX], int m, int n, char X[], char Y[]) {

printf("\n Cost Matrix with Directions:\n\n ");

for (int j = 0; j <= n; j++) {

if (j == 0) printf(" ");

else printf(" %c ", Y[j-1]);

}

printf("\n");

for (int i = 0; i <= m; i++) {

if (i == 0) printf(" ");

else printf(" %c ", X[i-1]);

for (int j = 0; j <= n; j++) {

printf("%2d%c ", dp[i][j], dir[i][j]);

}

printf("\n");

}

}

void printLCS(char X[], char Y[], int m, int n, int dp[MAX][MAX]) {

int i = m, j = n;

char lcs[MAX];

int index = dp[m][n];

lcs[index] = '\0';

while (i > 0 && j > 0) {

if (X[i - 1] == Y[j - 1]) {

lcs[index - 1] = X[i - 1];

i--;

j--;

index--;

}

else if (dp[i - 1][j] > dp[i][j - 1]) {

i--;

}

else {

j--;

}

}

printf("\nLongest Common Subsequence: %s\n", lcs);

}

int LCS\_with\_path(char X[], char Y[], int m, int n) {

int dp[MAX][MAX];

char dir[MAX][MAX];

int i, j;

for (i = 0; i <= m; i++) {

for (j = 0; j <= n; j++) {

dp[i][j] = 0;

dir[i][j] = ' ';

}

}

for (i = 1; i <= m; i++) {

for (j = 1; j <= n; j++) {

if (X[i - 1] == Y[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

dir[i][j] = '\\';

} else {

if (dp[i - 1][j] >= dp[i][j - 1]) {

dp[i][j] = dp[i - 1][j];

dir[i][j] = '|';

} else {

dp[i][j] = dp[i][j - 1];

dir[i][j] = '-';

}

}

}

}

printf("\nLength of LCS: %d\n", dp[m][n]);

printMatrix(dp, dir, m, n, X, Y);

printLCS(X, Y, m, n, dp);

return dp[m][n];

}

int main() {

char X[] = "AGCCCTAAGGGCTACCTAGCTT";

char Y[] = "GACAGCCTACAAGCGTTAGCTTG";

int m = strlen(X);

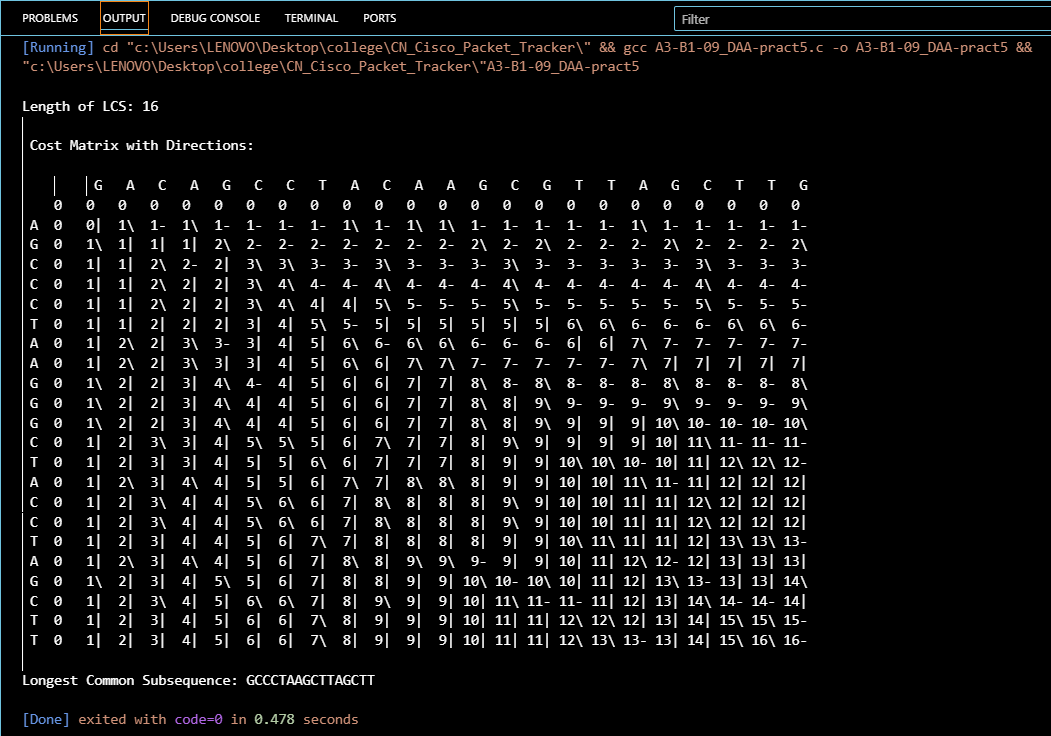
int n = strlen(Y);

LCS\_with\_path(X, Y, m, n);

return 0;

}

Output-



**TASK-2:** Find the longest repeating subsequence (LRS). Consider it as a variation of the

longest common subsequence (LCS) problem.

Let the given string be S. You need to find the LRS within S. To use the LCS framework, you effectively compare S with itself. So, consider string1 = S and string2 = S.

Example:

AABCBDC

LRS= ABC or ABD

Code-

#include <stdio.h>

#include <string.h>

char lrs\_results[100][50];

int lrs\_count = 0;

void find\_all\_lrs(char\* str, int i, int j, char\* current\_lrs, int current\_len, int dp[][100]) {

if (i == 0 || j == 0) {

current\_lrs[current\_len] = '\0';

strrev(current\_lrs);

int is\_duplicate = 0;

for (int k = 0; k < lrs\_count; k++) {

if (strcmp(lrs\_results[k], current\_lrs) == 0) {

is\_duplicate = 1;

break;

}

}

if (!is\_duplicate) {

strcpy(lrs\_results[lrs\_count], current\_lrs);

lrs\_count++;

}

return;

}

if (str[i - 1] == str[j - 1] && i != j) {

current\_lrs[current\_len] = str[i - 1];

find\_all\_lrs(str, i - 1, j - 1, current\_lrs, current\_len + 1, dp);

} else {

if (dp[i - 1][j] > dp[i][j - 1]) {

find\_all\_lrs(str, i - 1, j, current\_lrs, current\_len, dp);

} else if (dp[i][j - 1] > dp[i - 1][j]) {

find\_all\_lrs(str, i, j - 1, current\_lrs, current\_len, dp);

} else {

find\_all\_lrs(str, i - 1, j, current\_lrs, current\_len, dp);

find\_all\_lrs(str, i, j - 1, current\_lrs, current\_len, dp);

}

}

}

int main() {

char str[] = "AABCBDC";

int n = strlen(str);

int dp[100][100];

for (int i = 0; i <= n; i++) {

for (int j = 0; j <= n; j++) {

dp[i][j] = 0;

}

}

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

if (str[i - 1] == str[j - 1] && i != j) {

dp[i][j] = 1 + dp[i - 1][j - 1];

} else {

dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];

}

}

}

printf("String: %s\n", str);

printf("Length of LRS: %d\n", dp[n][n]);

char lrs\_str[50];

find\_all\_lrs(str, n, n, lrs\_str, 0, dp);

printf("All possible LRSs:\n");

for (int i = 0; i < lrs\_count; i++) {

printf("%s\n", lrs\_results[i]);

}

return 0;

}

Output-

